(Quick Revision)

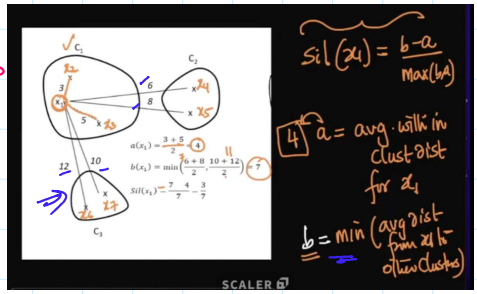
① K Means Algo *(Lloyds) → Random Initialization → Assigning the points to cluster → Update the centroid

* How to find the Best "K" → WCSS, Dunn Index, Elbow Method

* Mathematical formulation

(K Means) Centroids C_i s.t → (I.C) max, Intra (min)

Silhoutte Score →



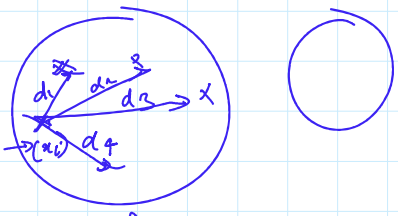
$x_1 \Rightarrow \frac{6+8}{2} = 7$, $a = 7$

$\Rightarrow \frac{12+10}{2} = 11$

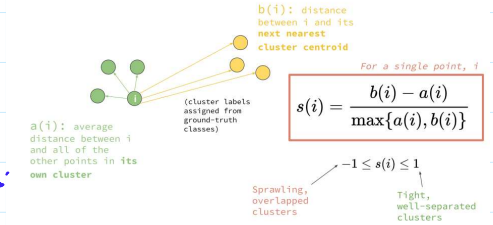
S.I = $\frac{(b-a)}{max(b,a)}$ → $[-1, 1]$ → (+1) good, (-1) bad cluster

a, and b

a ⇒

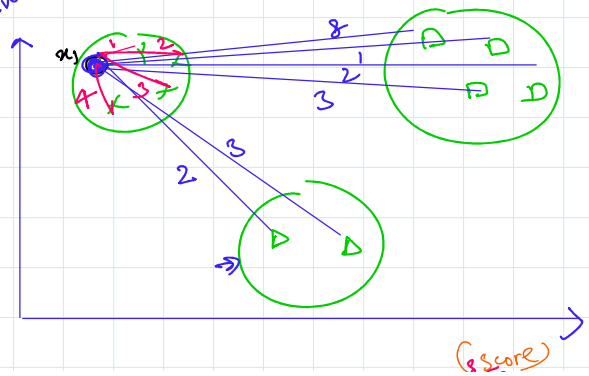


a for x_i = $\frac{\Sigma d}{n}$ → $(d_1+d_2+d_3+d_4)$ / 4



$a_{x_1} = \frac{1+2+3+4}{4}$

b = intercluster
a = intra



b = min (intercluster)

$b_{x_1} = \min\left[\frac{2+5}{2}, \frac{3+2+1+8}{4}\right]$

$(b_{x_1} = \min[3.5, (\ )]$

S.score for all data points = Avg (all the data points)

(s score)

$SI = \frac{b-a}{\max(b,a)}$ → index

a = 0.2, b = 0.8

$SI = \frac{0.6}{0.8} = \frac{3}{4}$

a = 0.8, b = 0.2

$S.I = \frac{0.2-0.8}{0.8} \Rightarrow -\frac{3}{4}$

(b to be more)

SI → b ↑ ⇒ good clustering

Silhoutte Score

- The silhouette score of a point **measures how close that point lies to its nearest neighbor points, across all clusters**.
- It provides information about clustering quality which can be used to determine whether further refinement by clustering should be performed on the current clustering.
- a is the the mean intra-cluster distance (i.e., mean distance to the other instances in the same cluster)
- b is the ~~nearest mean~~ inter-cluster distance (i.e., the mean distance to the instances of the next closest cluster). It is defined such that the instance's own cluster is excluding.

(The result of K Means Algo {centroids} is dependent on initial centroids)

(K Means is initialisation dependent)

@ K=3

INITIALISATION PROBLEM IN K MEANS

The choice of initial centroids can significantly impact the final clustering result because k-means is sensitive to the initial configuration of centroids. Poor initialization may lead to suboptimal clustering or convergence to local optima instead of the global optimum

Because of its sensitivity to initialization, k-means often produces different clustering results when run multiple times with different initializations. This variability makes it challenging to determine the best clustering solution objectively.

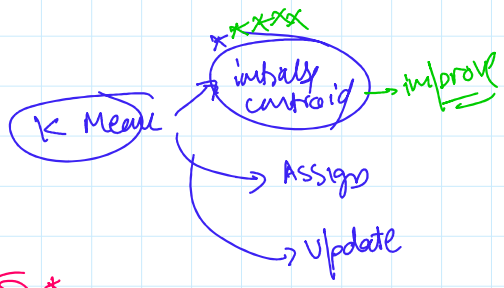
K Means ++

Problem with Initialisation

Run the algo multiple times & then check the metrics → wcss, D-I, S.I (such

check the wcss, D-I

→ # Check the wcss, D-1



Try some other Algo

(#) *

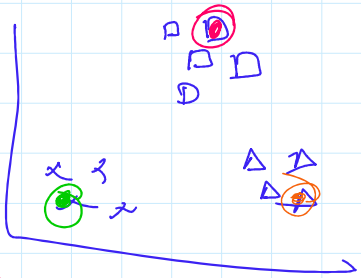
K Means (++) → It improves on the initialisation part

↳ solution for first part

(* Initialise centroids far away)

*

Probabilistic way → the prob. of choosing a point as cluster depends on the distance



K-Means ++ → Probabilistic nature → Smart initialisation

Ques → value = [3, 10, 15, 25, 35]

Prob = [7% 15% 20% 23% 35%]

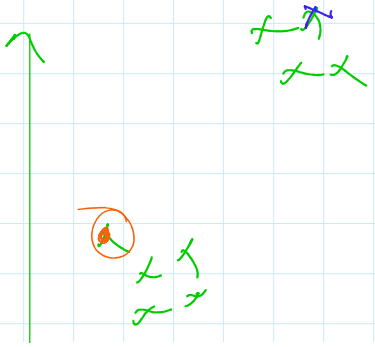
K Means ++ →

① initialise Randomly *

② Assign each point to nearest clusters

③ Update

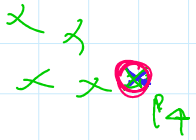
(#) Prob of selection a cluster ∝ (distance)^2 from nearest clusters

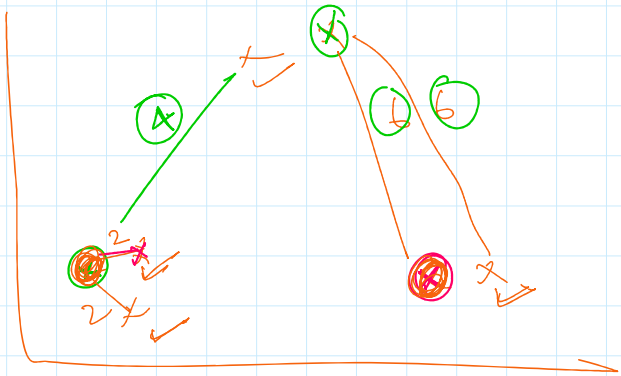


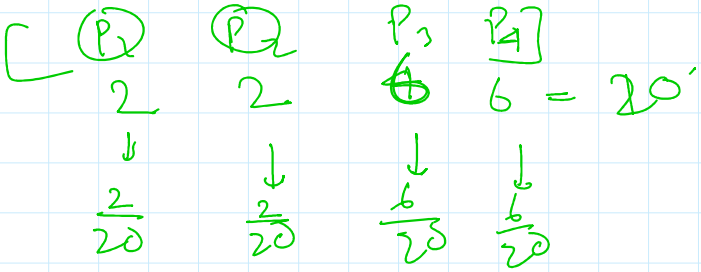
P1 P2 P3 P4

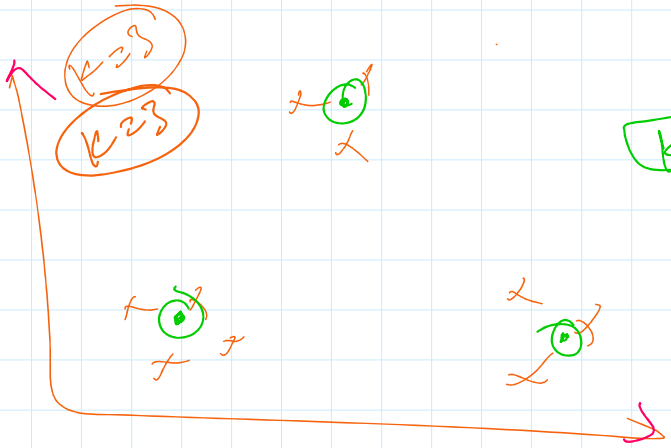
[10, 20, 30, 40] = 100

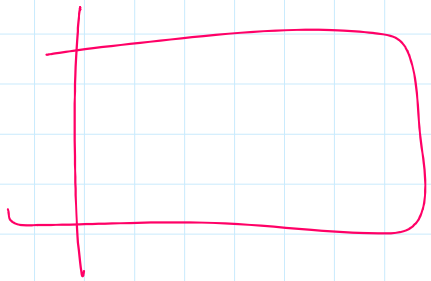
[10% 20% 30% 40%]



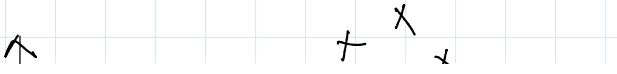


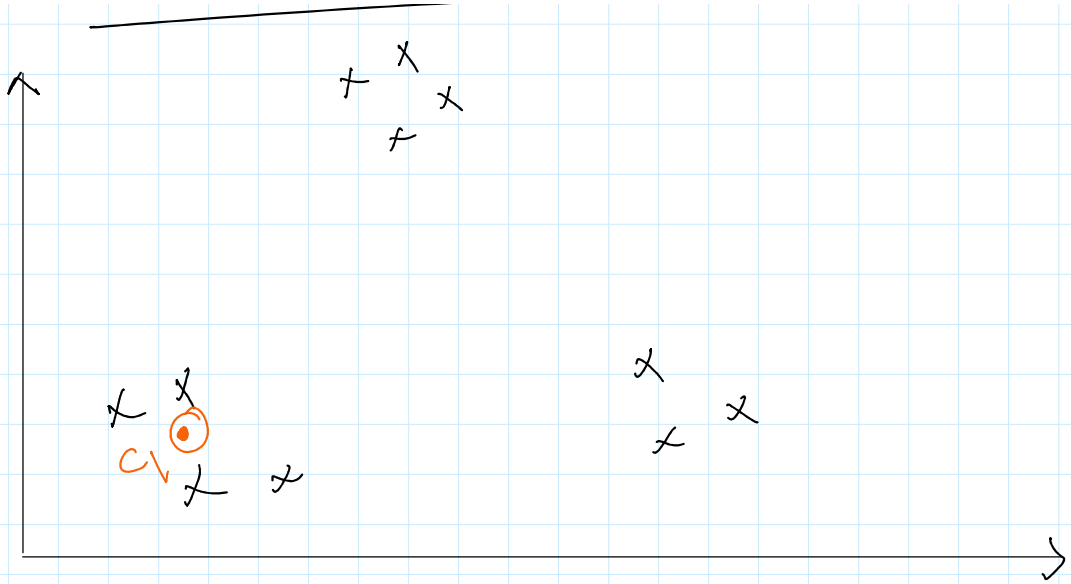






Could back at 10:27pm





Choosing Centroids in K Means ++

Step 1: Choose the First Centroid

- Randomly select the first centroid c_1 from the dataset points. This is done just like in standard K-means, ensuring that every data point is equally likely to be chosen as the initial centroid.

Step 2: Calculate Distances for Remaining Points

- For each data point x in the dataset, calculate the distance $D(x)$ from x to the nearest chosen centroid. Since there's only one centroid at this stage, $D(x)$ will be the distance from x to c_1.

Step 3: Choose the Second Centroid

- Choose the next centroid c_2 from the remaining data points, where the probability of choosing point x as the next centroid is proportional to $D(x)^2$. This means points further away from the first centroid are more likely to be selected as the second centroid.

Step 4: Update Distances

- After selecting c_2, update $D(x)$ for each data point x in the dataset. Now, $D(x)$ will be the distance to the nearest centroid, which could be either c_1 or c_2.

Step 5: Choose the Third Centroid

- Again, choose the next centroid c_3 from the remaining data points, with the probability of choosing point x as the next centroid being proportional to $D(x)^2$, based on the updated distances.

Step 6: Update Distances Again

- After selecting c_3, update $D(x)$ for each data point x, considering the nearest centroid among c_1, c_2, and c_3.

Step 7: Choose the Fourth Centroid

- Choose the fourth centroid c_4 using the same probabilistic method, based on the distances $D(x)$ to the nearest of the already chosen centroids c_1, c_2, and c_3.

K MEANS VS K MEANS ++

K-means Algorithm:

1. Initialization: Randomly select K points as the initial centroids from the dataset.
2. Assignment Step: Assign each data point to the nearest centroid based on the distance metric (usually Euclidean distance), forming K clusters.
3. Update Step: Recalculate the centroids as the mean of all points in each cluster.
4. Repeat: Repeat the assignment and update steps until the centroids no longer change significantly, indicating convergence.

K-means++ Algorithm:

1. Smart Initialization:
   - Randomly select the first centroid from the data points.
   - For each data point, compute the distance from the point to the nearest, already chosen centroid.
   - Select the next centroid from the data points with a probability proportional to the square of the distance to the nearest existing centroid. This step increases the chances of spreading out the initial centroids.
   - Repeat the above step until K centroids are chosen.
2. Assignment Step: Same as K-means.
3. Update Step: Same as K-means.
4. Repeat: Same as K-means.

python — Copy code

```python
from sklearn.cluster import KMeans

# Define the KMeans clustering model
kmeans = KMeans(
    n_clusters=3,   # Number of clusters
    init='k-means++',  # Initialization method ('k-means++' for smart
    n_init=10,      # Number of time the algorithm will run with differ
    max_iter=300,   # Maximum number of iterations of the algorithm for
    tol=1e-4,       # Relative tolerance with regards to inertia to dec
    random_state=42 # Determines random number generation for centroid
)

# Fit the model to the data
kmeans.fit(X)  # X is your data matrix or DataFrame

# Predict the closest cluster each sample in X belongs to
labels = kmeans.predict(X)
```

e

Explanation of Hyperparameters:

- n_clusters: The number of clusters to form and the number of centroids to generate. In the K-means++ context, this would be the number of centroids spread out through the smart initialization process.
- init: Method for initialization. 'k-means++' is the smart initialization method we've discussed, which spreads out the initial centroids. The alternative is 'random' for random initialization, or you can pass an ndarray to specify initial centers.
- n_init: Number of time the K-means algorithm will be run with different centroid seeds. The final results will be the best output of n_init consecutive runs in terms of inertia.
- max_iter: Maximum number of iterations of the K-means algorithm for a single run. This is a stopping criterion to ensure the algorithm terminates if it doesn't converge before this number of iterations.
- tol: Tolerance to declare convergence. It's the relative tolerance with regards to the difference in the value of the inertia (sum of squared distances of samples to their closest cluster center) to declare convergence. If the change in inertia is less than this tolerance, the algorithm stops.
- random_state: Determines random number generation for centroid initialization and can make the output of the algorithm deterministic. This is useful for reproducibility.

LIMITATIONS TO K Means & KMeans ++.

Sensitivity to Initial Centroids:

- K-means: Highly sensitive to the initial placement of centroids, which can lead to suboptimal clustering. Random initialization can result in different results on different runs.
- K-means++: Designed to mitigate this issue by spreading out initial centroids, but it still does not guarantee a global optimum.

2. Number of Clusters:

- Both algorithms require the number of clusters (K) to be specified in advance, which is not always practical or intuitive, especially when the structure of the data is unknown.

3. Cluster Shape and Size:

- Both algorithms assume that clusters are spherical and roughly of the same size, which makes them perform poorly on datasets with complex shapes or widely varying densities.

4. Outliers:

- K-means and K-means++ are sensitive to outliers. Outliers can significantly skew the centroids of the clusters, leading to incorrect clustering.

5. Convergence to Local Minima:

- K-means: Can easily converge to a local minimum, especially with poor initialization. This might not be the best possible solution.
- K-means++: Less prone to local minima due to smarter initialization, but it is still not immune to this issue.

6. Scalability:

- While generally efficient, both algorithms can struggle with very large datasets due to the need to compute distances between each point and each centroid at every iteration.

7. Categorical Data:

- Both algorithms are inherently designed for numerical data and do not naturally handle categorical variables. Special adaptations or pre-processing (like one-hot encoding) are needed, which might not always be effective.

8. Feature Space:

- Performance can degrade in high-dimensional spaces due to the "curse of dimensionality," where distance measures become less meaningful.

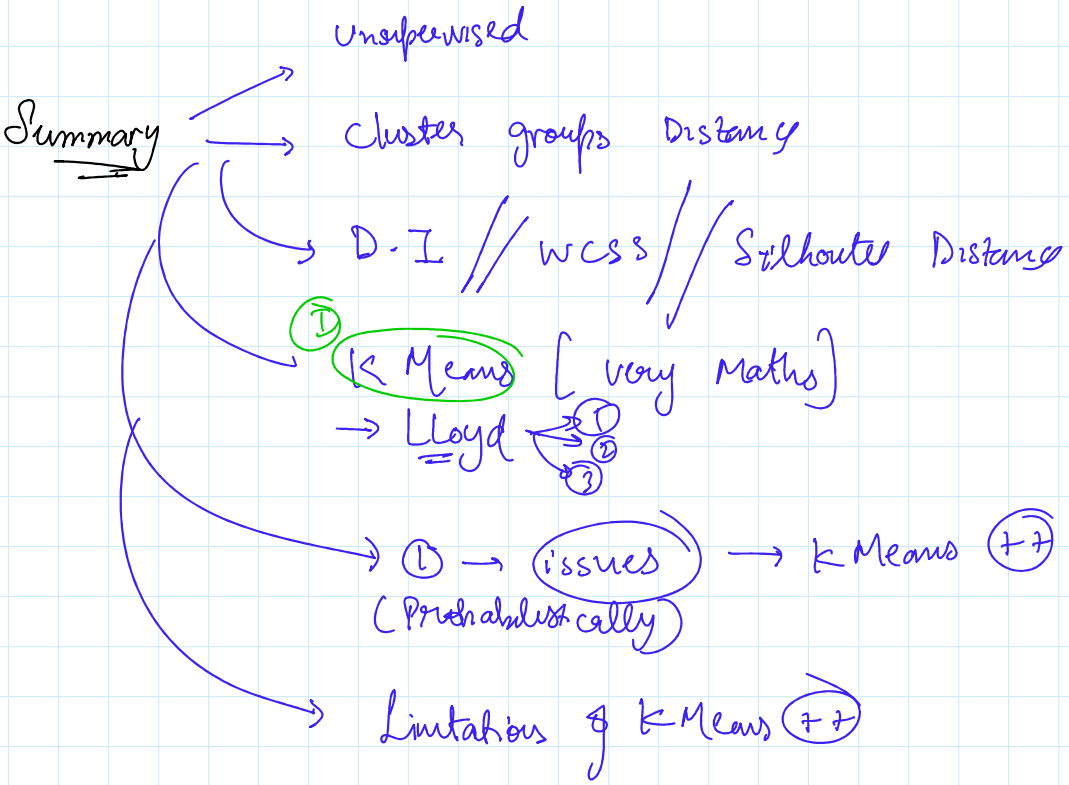
9. Deterministic Output:

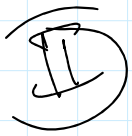
- K-means: The final output can vary between runs due to random initialization.
- K-means++: Although the initialization is smarter, the final output can still be influenced by the stochastic nature of the initialization process.

Mitigating Limitations:

- Pre-processing: Normalization, dimensionality reduction (e.g., PCA), and outlier removal can sometimes improve performance.
- Alternative Algorithms: Depending on the dataset, other clustering algorithms like DBSCAN, which does not require specifying the number of clusters and can handle arbitrary shapes, or hierarchical clustering might be more suitable.
- Multiple Runs: Running K-means multiple times with different initializations and choosing the best result based on a criterion like silhouette score can

sometimes mitigate the effects of poor initial centroids.

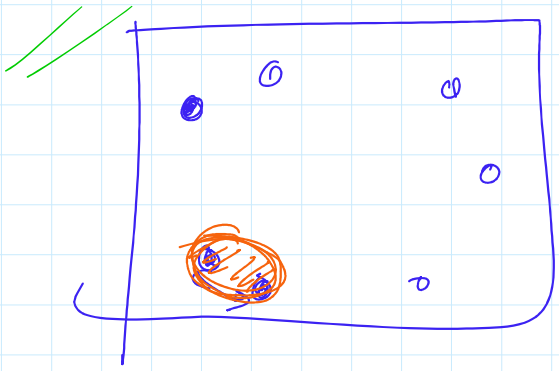


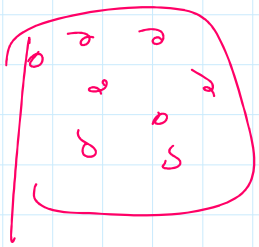


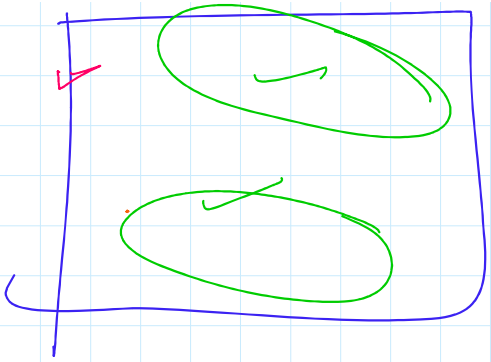
Hierarchical Clustering

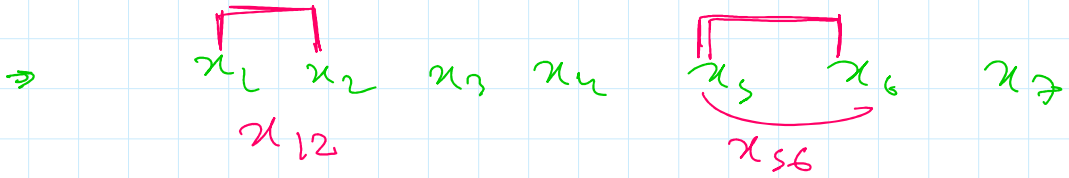
1. K = ? (Algo will still work)

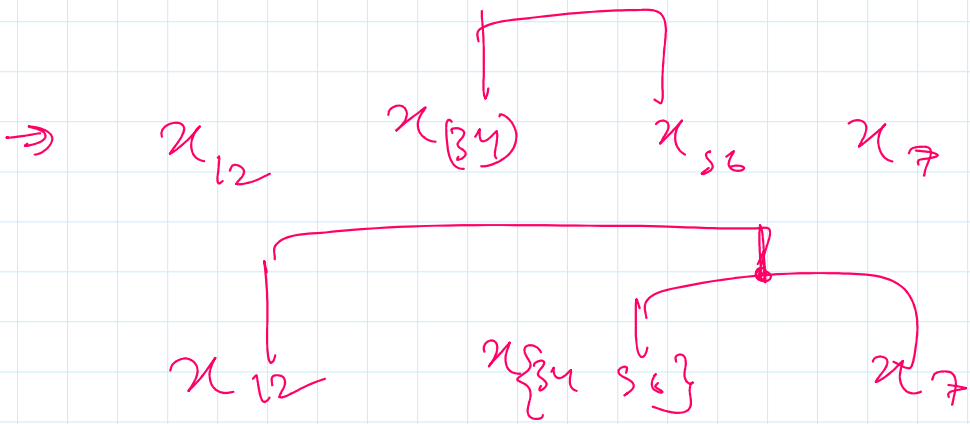
- Agglomerative (Adding) → Bottom Up
- Divisive

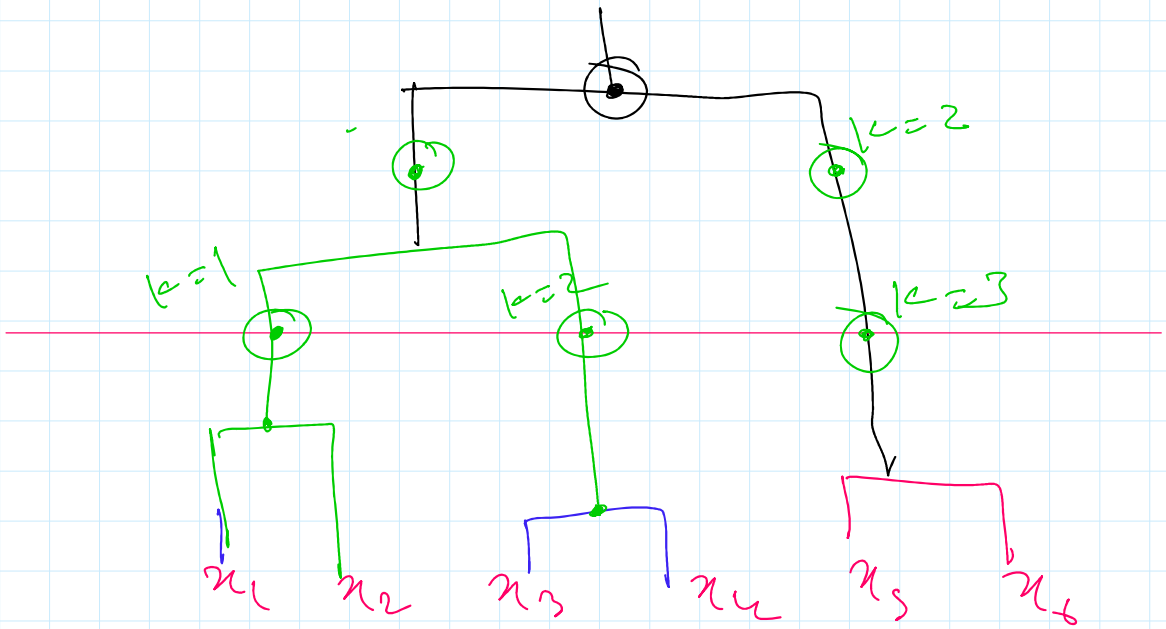







x_1 x_2 x_3 x_4 x_5 x_6 x_7
x_{12}
x_{56}


x_{12} $x_{(34)}$ x_{56} x_7
x_{12} $x_{\{34\,56\}}$ x_7


k=1
k=2
k=3
k=2
x_1 x_2 x_3 x_4 x_5 x_6